

LEARNING TO ACT BY PREDICTING THE FUTURE

Alexey Dosovitskiy

Intel Labs and University of Freiburg

Vladlen Koltun

Intel Labs

ABSTRACT

We present an approach to sensorimotor control in immersive environments. Our approach utilizes a high-dimensional sensory stream and a lower-dimensional measurement stream. The cotemporal structure of these streams provides a rich supervisory signal, which enables training a sensorimotor control model by interacting with the environment. The model is trained using supervised learning techniques, but without extraneous supervision. It learns to act based on raw sensory input from a complex three-dimensional environment. The presented formulation allows changing the agent’s goal at test time. We conduct extensive experiments in three-dimensional simulations based on the classical first-person game Doom. The results demonstrate that the presented approach outperforms sophisticated prior formulations, particularly on challenging tasks. The results also show that trained models successfully generalize across environments and goals. A model trained using the presented approach won the Full Deathmatch track of the Visual Doom AI Competition, which was held in previously unseen environments.

1 INTRODUCTION

Machine learning problems are commonly divided into three classes: supervised, unsupervised, and reinforcement learning. In this view, supervised learning is concerned with learning input-output mappings, unsupervised learning aims to find hidden structure in data, and reinforcement learning deals with goal-directed behavior (Murphy, 2012). Reinforcement learning is compelling because it considers the natural setting of an organism acting in its environment. It is generally taken to comprise a class of problems (learning to act), the mathematical formalization of these problems (maximizing the expected discounted return), and a family of algorithmic approaches (optimizing an objective derived from the Bellman equation) (Kaelbling et al., 1996; Sutton & Barto, 2017).

While reinforcement learning (RL) has achieved significant progress (Mnih et al., 2015), key challenges remain. One is sensorimotor control from raw sensory input in complex and dynamic three-dimensional environments, learned directly from experience. Another is the acquisition of general skills that can be flexibly deployed to accomplish a multitude of dynamically specified goals (Lake et al., 2016).

In this work, we propose an approach to sensorimotor control that aims to assist progress towards meeting these challenges. Our approach departs from the reward-based formalization commonly used in RL. Instead of a monolithic state and a scalar reward, we consider a stream of sensory input $\{\mathbf{s}_t\}$ and a stream of measurements $\{\mathbf{m}_t\}$. The sensory stream is typically high-dimensional and may include the raw visual, auditory, and tactile input. The measurement stream has lower dimensionality and constitutes a set of data that pertain to the agent’s current state. In a physical system, measurements can include attitude, supply levels, and structural integrity. In a three-dimensional computer game, they can include health, ammunition levels, and the number of adversaries overcome.

Our guiding observation is that the interlocked temporal structure of the sensory and measurement streams provides a rich supervisory signal. Given present sensory input, measurements, and goal, the agent can be trained to predict the effect of different actions on future measurements. Assuming that the goal can be expressed in terms of future measurements, predicting these provides all the information necessary to support action. This reduces sensorimotor control to supervised learning, while supporting learning from raw experience and without extraneous data. Supervision is pro-

vided by experience itself: by acting and observing the effects of different actions in the context of changing sensory inputs and goals.

This approach has two significant benefits. First, in contrast to an occasional scalar reward assumed in traditional RL, the measurement stream provides rich and temporally dense supervision that can stabilize and accelerate training. While a sparse scalar reward may be the only feedback available in a board game (Tesauro, 1994; Silver et al., 2016), a multidimensional stream of sensations is a more appropriate model for an organism that is learning to function in an immersive environment (Adolph & Berger, 2006).

The second advantage of the presented formulation is that it supports changing the agent’s goal at test time. Assuming that the goal can be expressed in terms of future measurements, the model can be trained to take the goal into account in its prediction of the future. At test time, the agent can predict future measurements given its current sensory input, measurements, and goal, and then simply select the action that best suits its present goal.

We evaluate the presented approach in immersive three-dimensional simulations that require visually navigating a complex three-dimensional environment, recognizing objects, and interacting with dynamic adversaries. We use the classical first-person game Doom, which introduced immersive three-dimensional games to popular culture (Kushner, 2003). The presented approach is given only raw visual input and the statistics shown to the player in the game, such as health and ammunition levels. No human gameplay is used, the model trains on raw experience.

Experimental results demonstrate that the presented approach outperforms state-of-the-art deep RL models, particularly on complex tasks. Experiments further demonstrate that models learned by the presented approach generalize across environments and goals, and that the use of vectorial measurements instead of a scalar reward is beneficial. A model trained with the presented approach won the Full Deathmatch track of the Visual Doom AI Competition, which took place in previously unseen environments. The presented approach outperformed the second best submission, which employed a substantially more complex model and additional supervision during training, by more than 50%.

2 BACKGROUND

The supervised learning (SL) perspective on learning to act by interacting with the environment dates back decades. Jordan & Rumelhart (1992) analyze this approach, review early work, and argue that the choice of SL versus RL should be guided by the characteristics of the environment. Their analysis suggests that RL may be more efficient when the environment provides only a sparse scalar reward signal, whereas SL can be advantageous when temporally dense multidimensional feedback is available.

Sutton (1988) analyzed temporal-difference (TD) learning and argued that it is preferable to SL for prediction problems in which the correctness of the prediction is revealed many steps after the prediction is made. Sutton’s influential analysis assumes a sparse scalar reward. TD and policy gradient methods have since come to dominate the study of sensorimotor learning (Kober et al., 2013; Mnih et al., 2015; Sutton & Barto, 2017). While the use of SL is natural in imitation learning (LeCun et al., 2005; Ross et al., 2013) or in conjunction with model-based RL (Levine & Koltun, 2013), the formulation of sensorimotor learning from raw experience as supervised learning is rare (Levine et al., 2016). Our work suggests that when the learner is exposed to dense multidimensional sensory feedback, direct future prediction can support effective sensorimotor coordination in complex dynamic environments.

Our approach has similarities to Monte Carlo methods. The convergence of such methods was analyzed early on and they were seen as theoretically advantageous, particularly when function approximators are used (Bertsekas, 1995; Sutton, 1995; Singh & Sutton, 1996). The choice of TD learning over Monte Carlo methods was argued on practical grounds, based on empirical performance on canonical examples (Sutton, 1995). While the understanding of the convergence of both types of methods has since improved (Szepesvári & Littman, 1999; Tsitsiklis, 2002; Even-Dar & Mansour, 2003), the argument for TD versus Monte Carlo is to this day empirical (Sutton & Barto, 2017). Sharp negative examples exist (Bertsekas, 2010). Our work deals with the more general setting of vectorial feedback and parameterized goals, and shows that a simple Monte-Carlo-type method performs extremely well in a compelling instantiation of this setting.

Vector-valued feedback has been considered in the context of multi-objective decision-making (Gábor et al., 1998; Roijers et al., 2013). Transfer across related tasks has been analyzed by Konidaris et al. (2012). Parameterized goals have been studied in the context of continuous motor skills such as throwing darts at a target (da Silva et al., 2012; Kober et al., 2012; Deisenroth et al., 2014). A general framework for sharing value function approximators across both states and goals has been described by Schaul et al. (2015). Our work is related, but develops a new model and shows that it achieves state-of-the-art performance in complex and dynamic three-dimensional environments.

Learning to act in simulated environments has been the focus of significant attention following the successful application of deep RL to Atari games by Mnih et al. (2015). A number of recent efforts applied related ideas to three-dimensional environments. Lillicrap et al. (2016) considered continuous and high-dimensional action spaces and learned control policies in the TORCS simulator. Mnih et al. (2016) described asynchronous variants of deep RL methods and demonstrated navigation in a three-dimensional labyrinth. Oh et al. (2016) augmented deep Q-networks with external memory and evaluated their performance on a set of tasks in Minecraft. In a recent technical report, Kulkarni et al. (2016b) proposed end-to-end training of successor representations and demonstrated navigation in the Doom game engine. In another recent report, Blundell et al. (2016) considered a nonparametric approach to control and conducted experiments in a three-dimensional labyrinth. We compare to state-of-the-art deep RL methods in Section 4 and demonstrate that the presented approach performs favorably.

Prediction of future states in dynamical systems was considered by Littman et al. (2001) and Singh et al. (2003). Predictive representations in the form of generalized value functions were advocated by Sutton et al. (2011). More recently, Oh et al. (2015) learned to predict future frames in Atari games. Prediction of full sensory input in realistic three-dimensional environments remains an open challenge, although significant progress is being made (Mathieu et al., 2016; Finn et al., 2016; Kalchbrenner et al., 2016). Our work considers prediction of future values of meaningful measurements from rich sensory input and shows that such prediction supports effective visuomotor control.

3 MODEL

Consider an agent that interacts with the environment over discrete time steps. At each time step t , the agent receives an observation \mathbf{o}_t and executes an action a_t based on this observation. We assume that the observations have the following structure: $\mathbf{o}_t = \langle \mathbf{s}_t, \mathbf{m}_t \rangle$, where \mathbf{s}_t is raw sensory input and \mathbf{m}_t is a set of measurements. In our experiments, \mathbf{s}_t is an image: the agent’s view of its three-dimensional environment. More generally, \mathbf{s}_t can include input from multiple sensory modalities. The measurements \mathbf{m}_t can indicate the attitude, supply levels, and structural integrity in a physical system, or health, ammunition, and score in a computer game.

The distinction between sensory input \mathbf{s}_t and measurements \mathbf{m}_t is somewhat artificial: both \mathbf{s}_t and \mathbf{m}_t constitute sensory input in different forms. In our model, the measurement vector \mathbf{m}_t is distinguished from other sensations in two ways. First, the measurement vector is the part of the observation that the agent will aim to predict. At present, predicting full sensory streams is beyond our capabilities (although see the work of Kalchbrenner et al. (2016) and van den Oord et al. (2016) for impressive recent progress). We therefore designate a subset of sensations as measurements that will be predicted. Second, we assume that the agent’s goal can be defined in terms of future measurements. Specifically, let τ_1, \dots, τ_n be a set of temporal offsets and let $\mathbf{f} = \langle \mathbf{m}_{t+\tau_1} - \mathbf{m}_t, \dots, \mathbf{m}_{t+\tau_n} - \mathbf{m}_t \rangle$ be the corresponding differences of future and present measurements. We assume that any goal that the agent will pursue can be defined as maximization of a function $u(\mathbf{f}; \mathbf{g})$. Any parametric function can be used. Our experiments use goals that are expressed as linear combinations of future measurements:

$$u(\mathbf{f}; \mathbf{g}) = \mathbf{g}^\top \mathbf{f}, \quad (1)$$

where the vector \mathbf{g} parameterizes the goal and has the same dimensionality as \mathbf{f} . This model generalizes the standard reinforcement learning formulation: the scalar reward signal can be viewed as a measurement, and exponential decay is one possible configuration of the goal vector.

To predict future measurements, we use a parameterized nonlinear function approximator, denoted by P :

$$\mathbf{p}_t^a = P(\mathbf{o}_t, a, \mathbf{g}; \boldsymbol{\theta}). \quad (2)$$

Here $a \in \mathcal{A}$ is an action, θ are the learned parameters of P , and \mathbf{p}_t^a is the resulting prediction. The dimensionality of \mathbf{p}_t^a matches the dimensionality of \mathbf{f} and \mathbf{g} . Note that the prediction is a function of the current observation, the considered action, and the goal. At test time, given learned parameters θ , the agent can choose the action that yields the best predicted outcome:

$$a_t = \arg \max_{a \in \mathcal{A}} \mathbf{g}^\top P(\mathbf{o}_t, a, \mathbf{g}; \theta). \quad (3)$$

The goal vector used at test time may or may not be identical to one of the goals used during training.

3.1 TRAINING

The predictor P is trained on experiences collected by the agent. No human trajectories are used. Starting with a random policy, the agent begins to interact with its environment. This interaction takes place over episodes that last for a fixed number of time steps or until a terminal event occurs.

Consider a set of experiences collected by the agent, yielding a set \mathcal{D} of training examples: $\mathcal{D} = \{\langle \mathbf{o}_i, a_i, \mathbf{g}_i, \mathbf{f}_i \rangle\}_{i=1}^N$. Here $\langle \mathbf{o}_i, a_i, \mathbf{g}_i \rangle$ is the input and \mathbf{f}_i is the output of example i . The predictor is trained using a regression loss:

$$\mathcal{L}(\theta) = \sum_{i=1}^N \|P(\mathbf{o}_i, a_i, \mathbf{g}_i; \theta) - \mathbf{f}_i\|^2. \quad (4)$$

A classification loss can be used for predicting categorical measurements, but this was not necessary in our experiments.

As the agent collects new experiences, the training set \mathcal{D} and the predictor used by the agent change. We maintain an experience memory of the M most recent experiences out of which a mini-batch of N examples is randomly sampled for every iteration of the solver. The parameters of the predictor used by the agent are updated after every k new experiences. Additional details are provided in Appendix B.

We have evaluated two training regimes:

1. Single goal: the goal vector is fixed throughout the training process.
2. Randomized goals: the goal vector for each episode is randomly sampled from a set of goals.

In both regimes, the agent follows an ε -greedy policy: it acts greedily according to the current goal with probability $1 - \varepsilon$, and selects a random action with probability ε . The value of ε is initially set to 1 and is decreased during training according to a fixed schedule.

3.2 ARCHITECTURE

The predictor P is a deep network parameterized by θ . The network architecture we use is shown in Figure 1. The network has three input modules: a perception module $S(\mathbf{s})$, a measurement module $M(\mathbf{m})$ and a goal module $G(\mathbf{g})$. In our experiments, \mathbf{s} is an image and the perception module S is implemented as a convolutional network. The measurement and goal modules are fully-connected networks. The outputs of the three input modules are concatenated, forming the joint input representation used for subsequent processing:

$$\mathbf{j} = J(\mathbf{s}, \mathbf{m}, \mathbf{g}) = \langle S(\mathbf{s}), M(\mathbf{m}), G(\mathbf{g}) \rangle. \quad (5)$$

Future measurements are predicted based on this input representation. The network emits predictions of future measurements for all actions at once. This could be done by a fully-connected module that absorbs the input representation and outputs predictions. However, we found that introducing additional structure into the prediction module enhances its ability to learn the fine differences between the outcomes of different actions. To this end, we build on the ideas of Wang et al. (2016) and split the prediction module into two streams: an expectation stream $E(\mathbf{j})$ and an action stream $A(\mathbf{j})$. The expectation stream predicts the average of the future measurements over all potential actions. The action stream concentrates on the fine differences between actions: $A(\mathbf{j}) = \langle A^1(\mathbf{j}), \dots, A^w(\mathbf{j}) \rangle$,

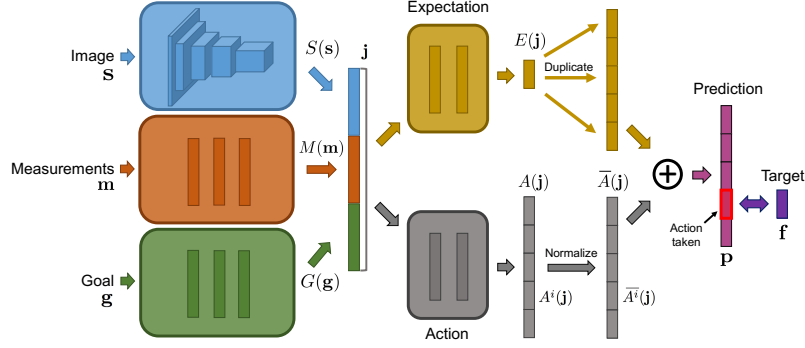


Figure 1: Network structure. The image s , measurements m , and goal g are first processed separately by three input modules. The outputs of these modules are concatenated into a joint representation j . This joint representation is processed by two parallel streams that predict the expected measurements $E(j)$ and the normalized action-conditional differences $\{\bar{A}^i(j)\}$, which are then combined to produce the final prediction for each action.

where $w = |\mathcal{A}|$ is the number of actions. We add a normalization layer at the end of the action stream that ensures that the average of the predictions of the action stream is zero for each future measurement:

$$\bar{A}^i(j) = A^i(j) - \frac{1}{w} \sum_{j=1}^w A^j(j) \quad (6)$$

for all i . The normalization layer subtracts the average over all actions from each prediction, forcing the expectation stream E to compensate by predicting these average values. The output of the expectation stream has dimensionality $\dim(\mathbf{f})$, where \mathbf{f} is the vector of future measurements. The output of the action stream has dimensionality $w \cdot \dim(\mathbf{f})$.

The output of the network is a prediction of future measurements for each action, composed by summing the output of the expectation stream and the normalized action-conditional output of the action stream:

$$\langle \bar{A}^1(j) + E(j), \dots, \bar{A}^w(j) + E(j) \rangle. \quad (7)$$

The output of the network has the same dimensionality as the output of the action stream.

4 EXPERIMENTS

We evaluate the presented approach in immersive three-dimensional simulations based on the classical game Doom. In these simulations, the agent has a first-person view of the environment and must act based on the same visual information that is shown to human players in the game. To interface with the game engine, we use the ViZDoom platform developed by Kempka et al. (2016). One of the advantages of this platform is that it allows running the simulation at thousands of frames per second on a single CPU core, which enables training models on tens of millions of simulation steps in a single day.

We compare the presented approach to state-of-the-art deep RL methods in four scenarios of increasing difficulty, study generalization across environments and goals, and evaluate the importance of different aspects of the model.

4.1 SETUP

Scenarios. We use four scenarios of increasing difficulty:

- D1 Gathering health kits in a square room. (“Basic”)
- D2 Gathering health kits and avoiding poison vials in a maze. (“Navigation”)
- D3 Defending against adversaries while gathering health and ammunition in a maze. (“Battle”)

D4 Defending against adversaries while gathering health and ammunition in a more complicated maze. (“Battle 2”)

These scenarios are illustrated in Appendix A and in the supplementary video (<http://bit.ly/2f9tacZ>).

The first two scenarios are provided with the ViZDoom platform. In D1, the agent is in a square room and its health is declining at a constant rate. To survive, it must move around and collect health kits, which are distributed abundantly in the room. This task is easy: as long as the agent learns to avoid walls and keep traversing the room, performance is good. In D2, the agent is in a maze and its health is again declining at a constant rate. Here it must again collect health kits that increase its health, but it must also avoid blue poison vials that decrease health. This task is harder: the agent must learn to traverse irregularly shaped passageways, and to distinguish health kits from poison vials. In both tasks, the agent has access to three binary sub-actions: move forward, turn left, and turn right. Any combination of these three can be used at any given time, resulting in 8 possible actions. The only measurement provided to the agent in these scenarios is health.

The last two scenarios, D3 and D4, are more challenging and were designed by us using elements of the ViZDoom platform. Here the agent is armed and is under attack by alien monsters. The monsters spawn abundantly, move around in the environment, and shoot fireballs at the agent. Health kits and ammunition are sporadically distributed throughout the environment and can be collected by the agent. The environment is a simple maze in D3 and a more complex one in D4. In both scenarios, the agent has access to eight sub-actions: move forward, move backward, turn left, turn right, strafe left, strafe right, run, and shoot. Any combination of these sub-actions can be used, resulting in 256 possible actions. The agent is provided with three measurements: health, ammunition, and frag count (number of monsters killed).

Models. We use two variants of the network architecture presented in Section 3. We refer to these variants as `shallow` and `deep`. Both follow the general structure shown in Figure 1, but differ in the exact configuration of the modules. The `shallow` version was configured to be as close as possible to the DQN model of Mnih et al. (2015), to ensure a fair comparison. This is the variant we use in all comparisons to prior work. The `deep` version has more layers but a much smaller number of parameters. This variant was used in experiments that do not involve prior work. Additional details are provided in Appendix B.

Training and testing. The agent is trained and tested over episodes. Each episode terminates after 525 steps (equivalent to 1 minute of real time) or when the agent’s health declines to zero. Reported measurements are averaged over the complete duration of each episode and across episodes.

4.2 RESULTS

Comparison to prior work. We have compared the presented approach to three deep RL methods: DQN (Mnih et al., 2015), A3C (Mnih et al., 2016), and DSR (Kulkarni et al., 2016b). DQN is a standard baseline for visuomotor control due to its impressive performance on Atari games. A3C is more recent and is commonly regarded as the state of the art in this area. DSR is described in a recent technical report and we included it because the authors also used the ViZDoom platform in experiments, albeit with a simple task. Further details on the setup of the prior approaches are provided in Appendix C.

The performance of the different approaches during training is shown in Figure 2. In reporting the results of these experiments, we refer to our approach as DFP (direct future prediction). For the first two scenarios, all approaches were trained to maximize health. For these scenarios, Figure 2 reports average health over the course of training. For the last two scenarios, all approaches were trained to maximize a linear combination of the three normalized measurements (ammo, health, and frags) with coefficients (0.5, 0.5, 1). For these scenarios, Figure 2 reports average frags. Each data point in the plots averages information from 20,000 steps of testing.

DQN, A3C, and DFP were trained for 50 million steps. The training procedure for DSR is much slower and can only process roughly 1 million simulation steps per day. For this reason, we were only able to evaluate DSR on the Basic scenario and were not able to perform extensive hyperparameter tuning. We report results for this technique after a week of training. (This week was sufficient

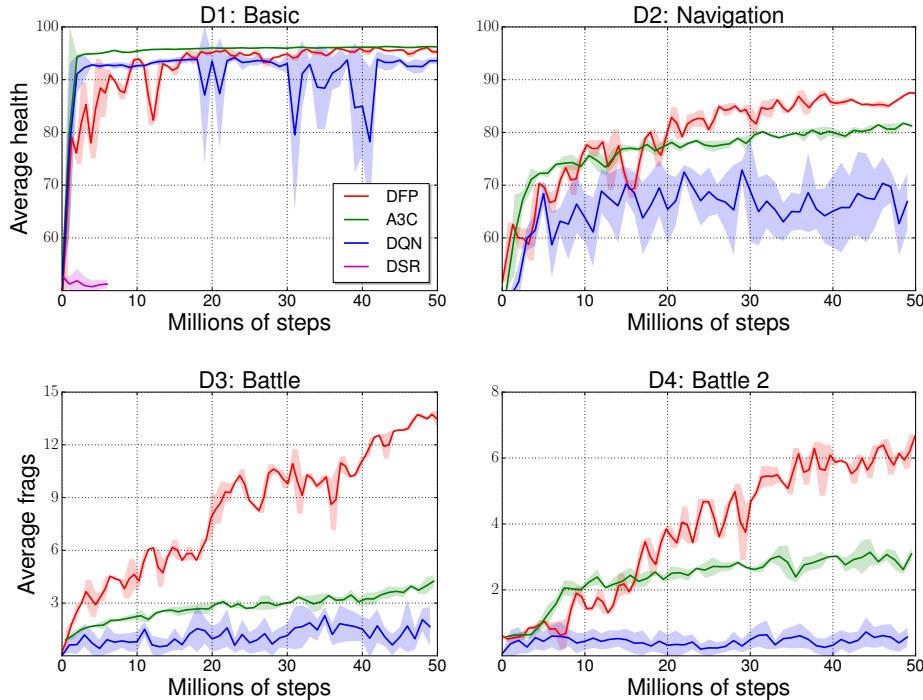


Figure 2: Performance of different approaches during training. DQN, A3C, and DFP achieve similar performance in the Basic scenario. DFP outperforms the prior approaches in the other three scenarios, with a multiplicative gap in performance in the most complex ones (D3 and D4).

to significantly exceed the number of training steps reported in the experiments of Kulkarni et al. (2016b), but not sufficient to approach the number of steps afforded by the other approaches.) For a fair comparison to prior work, DFP used the shallow version of our model.

Table 1 reports the performance of the models after training. Each fully trained model was tested over 1 million simulation steps. (Consecutive episodes were automatically launched until the stated number of steps was reached for each method.) The table reports average health for scenarios D1 and D2, and average frags for D3 and D4. We also report the training speed for each approach, in millions of simulation steps per day of training. The performance of the different models is additionally illustrated in the supplementary video.

	D1	D2	D3	D4	steps/day
DQN	94.7	70.0	2.5	0.2	6M
A3C	96.0	81.2	3.8	3.0	50M
DSR	51.7	—	—	—	1M
DFP	95.8	86.1	13.8	6.6	70M

Table 1: Comparison to prior work.

The performance of DQN, A3C, and DFP is similar in the Basic scenario. As reported in Table 1, all three approaches achieve average health of around 95% during testing, with virtually identical performance by A3C and DFP. In the more complex Navigation scenario, a significant gap opens up between DQN and A3C; this is consistent with the experiments of Mnih et al. (2016). DFP achieves the best performance in this scenario, with a 5 percentage point advantage during testing. Note that in these first two scenarios DFP was only given a single measurement per time step, health.

The performance of DQN, A3C, and DFP is similar in the Basic scenario. As reported in Table 1, all three approaches achieve average health of around 95% during testing, with virtually identical performance by A3C and DFP. In the more complex Navigation scenario, a significant gap opens up between DQN and A3C; this is consistent with the experiments of Mnih et al. (2016). DFP achieves the best performance in this scenario, with a 5 percentage point advantage during testing. Note that in these first two scenarios DFP was only given a single measurement per time step, health.

In the more complex Battle and Battle 2 scenarios (D3 and D4), DFP dominates the other approaches. It outperforms A3C at test time by a factor of 3.6 in D3 and by a factor of 2.2 in D4. Note that the advantage of DFP is particularly significant in the scenarios that provide richer measurements: three measurements per time step in D3 and D4. The effect of multiple measurements is further evaluated in controlled experiments reported below.

Generalization across environments. We now evaluate how the behaviors learned by the presented approach generalize across different environments. To this end, we have created 100 ran-

domly textured versions of the mazes from scenarios D3 and D4. We used 90 of these for training and 10 for testing, with disjoint sets of textures in the training and testing environments. We call these scenarios D3-tx and D4-tx.

Table 2 shows the performance of the approach for different combinations of training and testing regimes. For example, the entry in the D4-tx row of the D3 column shows the performance (in average number of frags) of a model trained in D3 and tested in D4-tx. Not surprisingly, a model trained in the simple D3 environment does not learn sufficient invariance to surface appearance to generalize well to other environments. Training in the more complex multi-texture environment in D4 yields better generalization: the trained model performs well in D3 and exhibits non-trivial performance in D3-tx and D4-tx. Finally, exposing the model to significant variation in surface appearance in D3-tx or D4-tx during training yields very good generalization.

		Train				
		D3	D4	D3-tx	D4-tx	D4-tx-L
Test	D3	14.2	9.5	15.4	9.8	11.6
	D4	1.1	7.2	3.0	3.8	6.0
	D3-tx	2.5	4.2	11.4	7.4	10.6
	D4-tx	1.3	2.5	3.6	4.2	6.4

Table 2: Generalization across environments.

The last column of Table 2 additionally reports the performance of a higher-capacity model trained in D4-tx. This model has the same architecture but each layer is wider by a factor of two. This combination is referred to as D4-tx-L. As shown in the table, this model performs even better.

Visual Doom AI Competition. To further evaluate the presented approach, we enrolled in the Visual Doom AI Competition, held during September 2016. The competition evaluated sensorimotor control models that act based on raw visual input. The competition had the form of a tournament: the submitted agents play multiple games against each other, their performance measured by aggregate frags. The competition included two tracks. The Limited Deathmatch track was held in a known environment that was given to the participants in advance at training time. The Full Deathmatch track evaluated generalization to previously unseen environments and took place in multiple new environments that were not available to the participating teams at training time. We only enrolled in the Full Deathmatch track. Our model was trained using a variant of the D4-tx-L regime.

Our model won, outperforming the second best submission by more than 50%. That submission, described by Lample & Chaplot (2016), constitutes a strong baseline. It is a deep recurrent Q-network that incorporates an LSTM and was trained using reward shaping and extra supervision from the game engine. Specifically, the authors took advantage of the ability provided by the ViZDoom platform to use the internal configuration of the game, including ground-truth knowledge of the presence of enemies in the field of view, during training. The authors’ report shows that this additional supervision improved performance significantly. Our model, which is simpler, achieved even higher performance without such additional supervision.

Goal-agnostic training. We now evaluate the ability of the presented approach to learn behaviors that adapt to varying goals at test time. These experiments are performed in the Battle scenario. We use three training regimes: (a) fixed goal vector during training, (b) random goal vector with each value sampled uniformly from $[0, 1]$ for every episode, (c) random goal vector with each value sampled uniformly from $[-1, 1]$ for every episode. Intuitively, the second regime assumes that the agent will be maximizing all of the measurements at test time, to unknown degrees. The third regime makes no assumptions as to whether a measured quantity will be desirable or not.

The results are shown in Table 3. Each column corresponds to a training regime and each row to a different test-time goal. Goals are given by the weights of the three measurements (ammo, health, and frags) four seconds into the future. The first test-time goal in Table 3 is the goal vector used in the battle scenarios in the prior experiments, the second seeks to hoard ammunition, the third is a pacifist (maximize ammo and health, minimize frags), the fourth seeks to aimlessly drain ammunition. Each cell in the table reports the average value of each of the three measurements when a model that was trained as specified by the column was then tested as specified by the row.

We draw two main conclusions. First, on the main task (first row), goal-agnostic training performs as well as goal-driven training or even slightly better. Without knowing the final goal in advance, the agent learns to perform the task as well as when the goal was known at training time. Second,

	fixed goal (0.5, 0.5, 1)			random goals [0, 1]			random goals [-1, 1]		
test goal	ammo	health	frags	ammo	health	frags	ammo	health	frags
(0.5, 0.5, 1)	39.6	97.1	13.9	59.1	97.7	15.7	40.7	96.2	13.1
(1, 0, 0)	29.4	79.5	0.1	52.4	85.0	0.1	51.0	89.1	0.0
(1, 1, -1)	22.5	82.0	0.0	10.5	75.6	0.1	48.9	93.9	0.0
(-1, 0, 0)	9.1	74.9	0.7	10.0	70.9	0.3	7.1	72.4	0.6

Table 3: Generalization across goals. Each group of three columns corresponds to a training regime, each row corresponds to a test-time goal. On the main task (first row), the approach performs well even without knowing the goal at training time. Furthermore, goal-agnostic training improves generalization across goals.

all agents, even the one trained with a fixed goal, generalize to new goals. Agents trained with randomized goals generalize better than the one trained with a fixed goal.

Ablation study. We now perform an ablation study using the D3-tx scenario. Specifically, we evaluate the importance of vectorial feedback versus scalar reward, and the effect of predicting measurements at multiple temporal offsets. The results are summarized in Table 4. The table reports the performance (in average frags at test time) of our full model (predicting three measurements at six temporal offsets) and of ablated variants that only predict frags (a scalar reward) and/or only predict at the farthest temporal offset. As the results demonstrate, predicting multiple measurements significantly improves the performance of the learned model, even when it is evaluated by only one of those measurements. Predicting measurements at multiple future times is also beneficial. This supports the intuition that a dense flow of multivariate measurements is a better training signal than a scalar reward.

	frags
all measurements, all offsets	11.4
all measurements, one offset	9.2
frags only, all offsets	5.4
frags only, one offset	4.1

Table 4: Ablation study.

5 DISCUSSION

We presented an approach to sensorimotor control in immersive environments. Our approach is simple and demonstrates that supervised learning techniques can be adapted to learning to act in complex and dynamic three-dimensional environments given raw sensory input and intrinsic measurements. The model trains on raw experience, by interacting with the environment without extraneous supervision. Natural supervision is provided by the cotemporal structure of the sensory and measurement streams. Our experiments have demonstrated that this simple approach outperforms sophisticated deep reinforcement learning formulations on challenging tasks in immersive environments. Experiments have further demonstrated that the use of multivariate measurements provides a significant advantage over conventional scalar rewards and that the trained model can effectively adapt to new goals.

The presented work can be extended in multiple ways that are important for broadening the range of behaviors that can be learned. First, the presented model is purely reactive: it acts based on the current frame only, with no explicit facilities for memory and no test-time retention of internal representations. Recent work has explored memory-based models (Oh et al., 2016) and integrating such ideas with the presented approach may yield substantial advances. Second, significant progress in behavioral sophistication will likely require temporal abstraction and hierarchical organization of learned skills (Barto & Mahadevan, 2003; Kulkarni et al., 2016a). Third, the presented model was developed for discrete action spaces; applying the presented ideas to continuous action spaces would be interesting (Lillicrap et al., 2016). Finally, predicting features learned directly from rich sensory input can blur the distinction between sensory and measurement streams (Mathieu et al., 2016; Finn et al., 2016; Kalchbrenner et al., 2016).

REFERENCES

- Karen E. Adolph and Sarah E. Berger. Motor development. In *Handbook of Child Psychology*, volume 2, pp. 161–213. Wiley, 6th edition, 2006.
- Andrew G. Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(1-2), 2003.
- Dimitri P. Bertsekas. A counterexample to temporal differences learning. *Neural Computation*, 7(2), 1995.
- Dimitri P. Bertsekas. Pathologies of temporal difference methods in approximate dynamic programming. In *IEEE Conference on Decision and Control*, 2010.
- Charles Blundell, Benigno Uria, Alexander Pritzel, Yazhe Li, Avraham Ruderman, Joel Z. Leibo, Jack Rae, Daan Wierstra, and Demis Hassabis. Model-free episodic control. *arXiv:1606.04460*, 2016.
- Bruno Castro da Silva, George Konidaris, and Andrew G. Barto. Learning parameterized skills. In *ICML*, 2012.
- Marc Peter Deisenroth, Peter Englert, Jan Peters, and Dieter Fox. Multi-task policy search for robotics. In *ICRA*, 2014.
- Eyal Even-Dar and Yishay Mansour. Learning rates for Q-learning. *JMLR*, 5, 2003.
- Chelsea Finn, Ian J. Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. In *NIPS*, 2016.
- Zoltán Gábor, Zsolt Kalmár, and Csaba Szepesvári. Multi-criteria reinforcement learning. In *ICML*, 1998.
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *ICCV*, 2015.
- Michael I. Jordan and David E. Rumelhart. Forward models: Supervised learning with a distal teacher. *Cognitive Science*, 16(3), 1992.
- Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *JAIR*, 4, 1996.
- Nal Kalchbrenner, Aaron van den Oord, Karen Simonyan, Ivo Danihelka, Oriol Vinyals, Alex Graves, and Koray Kavukcuoglu. Video pixel networks. *arXiv:1610.00527*, 2016.
- Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. ViZDoom: A Doom-based AI research platform for visual reinforcement learning. In *IEEE Conference on Computational Intelligence and Games*, 2016.
- Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.
- Jens Kober, Andreas Wilhelm, Erhan Oztop, and Jan Peters. Reinforcement learning to adjust parametrized motor primitives to new situations. *Autonomous Robots*, 33(4), 2012.
- Jens Kober, J. Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *IJRR*, 32(11), 2013.
- George Konidaris, Ilya Scheidwasser, and Andrew G. Barto. Transfer in reinforcement learning via shared features. *JMLR*, 13, 2012.
- Tejas D. Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Joshua B. Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *NIPS*, 2016a.
- Tejas D. Kulkarni, Ardavan Saeedi, Simanta Gautam, and Samuel J. Gershman. Deep successor reinforcement learning. *arXiv:1606.02396*, 2016b.
- David Kushner. *Masters of Doom: How Two Guys Created an Empire and Transformed Pop Culture*. Random House, 2003.
- Brenden M. Lake, Tomer D. Ullman, Joshua B. Tenenbaum, and Samuel J. Gershman. Building machines that learn and think like people. *arXiv:1604.00289*, 2016.
- Guillaume Lample and Devendra Singh Chaplot. Playing FPS games with deep reinforcement learning. *arXiv:1609.05521*, 2016.
- Yann LeCun, Urs Muller, Jan Ben, Eric Cosatto, and Beat Flepp. Off-road obstacle avoidance through end-to-end learning. In *NIPS*, 2005.
- Sergey Levine and Vladlen Koltun. Guided policy search. In *ICML*, 2013.

- Sergey Levine, Peter Pastor, Alex Krizhevsky, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. In *ISER*, 2016.
- Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *ICLR*, 2016.
- Michael L. Littman, Richard S. Sutton, and Satinder P. Singh. Predictive representations of state. In *NIPS*, 2001.
- Michaël Mathieu, Camille Couprie, and Yann LeCun. Deep multi-scale video prediction beyond mean square error. In *ICLR*, 2016.
- Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540), 2015.
- Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, 2016.
- Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.
- Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L. Lewis, and Satinder P. Singh. Action-conditional video prediction using deep networks in Atari games. In *NIPS*, 2015.
- Junhyuk Oh, Valliappa Chockalingam, Satinder P. Singh, and Honglak Lee. Control of memory, active perception, and action in Minecraft. In *ICML*, 2016.
- Diederik M. Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. A survey of multi-objective sequential decision-making. *JAIR*, 48, 2013.
- Stéphane Ross, Narek Melik-Barkhudarov, Kumar Shaurya Shankar, Andreas Wendel, Debadeepta Dey, J. Andrew Bagnell, and Martial Hebert. Learning monocular reactive UAV control in cluttered natural environments. In *ICRA*, 2013.
- Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *ICML*, 2015.
- David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587), 2016.
- Satinder P. Singh and Richard S. Sutton. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22(1-3), 1996.
- Satinder P. Singh, Michael L. Littman, Nicholas K. Jong, David Pardoe, and Peter Stone. Learning predictive state representations. In *ICML*, 2003.
- Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3, 1988.
- Richard S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *NIPS*, 1995.
- Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition, 2017.
- Richard S. Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M. Pilarski, Adam White, and Doina Precup. Horde: a scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *AAMAS*, 2011.
- Csaba Szepesvári and Michael L. Littman. A unified analysis of value-function-based reinforcement learning algorithms. *Neural Computation*, 11(8), 1999.
- Gerald Tesauro. TD-gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, 6(2), 1994.
- John N. Tsitsiklis. On the convergence of optimistic policy iteration. *JMLR*, 2002.
- Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. WaveNet: A generative model for raw audio. *arXiv:1609.03499*, 2016.
- Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning. In *ICML*, 2016.

A SCENARIOS

Figure A1 illustrates the four scenarios used in our experiments. Additional illustration is provided in the supplementary video (<http://bit.ly/2f9tacZ>).

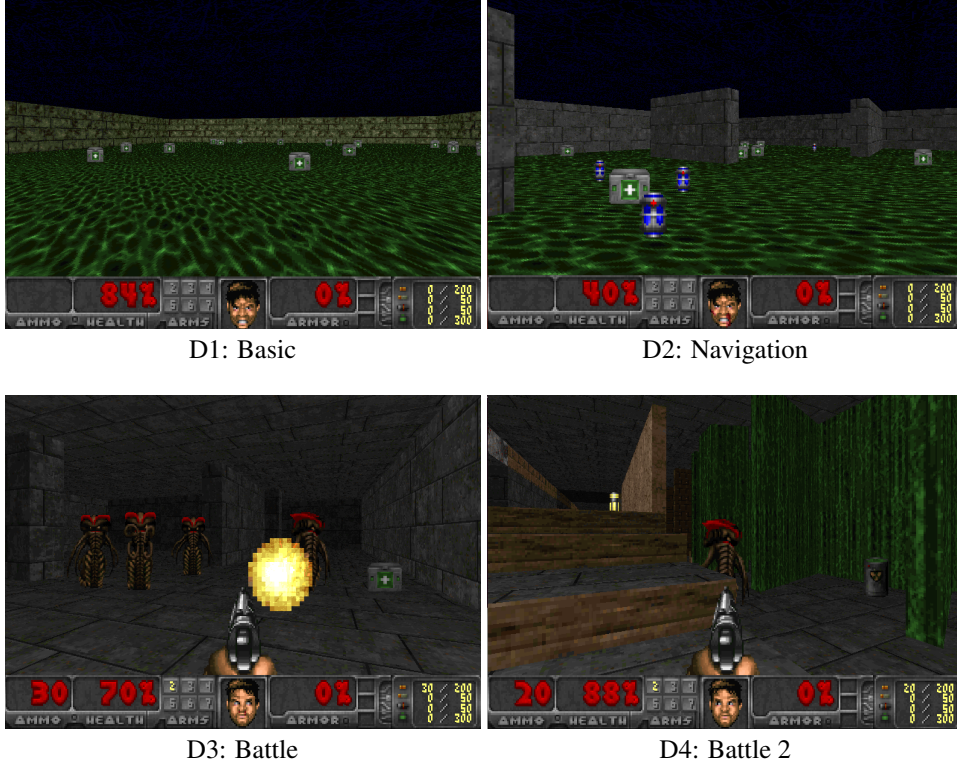


Figure A1: Example frames from the four scenarios.

B IMPLEMENTATION DETAILS

B.1 NETWORK ARCHITECTURES

The detailed architectures of two network variants – shallow and deep – are shown in Tables B2 and B3. The `shallow` network follows the architecture of Mnih et al. (2015) as closely as possible. The `deep` one has more layers, but each layer is much narrower. The `large` one is the same as the `deep` one, but all layers are wider by a factor of two. The `large` and `deep` variants have fewer parameters than the `shallow` one: approximately 0.8 million in the `deep` network and 2.0 million in the `large` network, as compared to 6.4 million in the `shallow` network. In all networks we use the leaky ReLU nonlinearity $\text{LReLU}(x) = x \cdot \delta[x \geq 0] + 0.2x \cdot \delta[x < 0]$ after each non-terminal layer. We initialized the weights of the network as proposed by He et al. (2015).

We empirically validate the architectural choices in the D3-tx regime. We compare the `shallow` and `deep` architectures, as well as three variants of the `deep` architecture:

- No split: no expectation/action split, simply predict future measurements with a fully-connected network.
- Late split: predict both the expectation and the action-conditional differences with a single module.
- Early split: predict the expectation and the action-conditional differences with two separate networks.

The results are reported in Table B1. The `deep` architecture performs slightly better than the `shallow` one, even though it has 8 times fewer parameters. All modifications of the `deep` architecture hurt performance, showing that (a) the two-stream formulation is beneficial, and (b) both too much and too little interaction between the two streams is detrimental to performance.

	<code>shallow</code>	<code>deep</code>	<code>no split</code>	<code>late split</code>	<code>early split</code>
Score	10.9	11.4	9.7	8.3	8.2

Table B1: Evaluation of different network architectures.

Module	Input dimension	Channels	Kernel	Stride
Perception	$84 \times 84 \times 1$	32	8	4
	$21 \times 21 \times 32$	64	4	2
	$10 \times 10 \times 64$	64	3	1
	$10 \cdot 10 \cdot 64$	512	—	—
Measurement	3	128	—	—
	128	128	—	—
	128	128	—	—
Goal	$3 \cdot 6$	128	—	—
	128	128	—	—
	128	128	—	—
Expectation	$512 + 128 + 128$	512	—	—
	512	$3 \cdot 6$	—	—
Action	$512 + 128 + 128$	512	—	—
	512	$3 \cdot 6 \cdot 256$	—	—

Table B2: The `shallow` architecture.

Module	Input dimension	Channels	Kernel	Stride
Perception	$160 \times 120 \times 1$	8	5	4
	$40 \times 30 \times 32$	16	3	2
	$20 \times 15 \times 64$	32	3	2
	$10 \times 8 \times 64$	64	3	2
	$5 \cdot 4 \cdot 64$	64	—	—
Measurement	3	64	—	—
	64	64	—	—
	64	64	—	—
Goal	$3 \cdot 6$	64	—	—
	64	64	—	—
	64	64	—	—
Expectation	$64 + 64 + 64$	128	—	—
	128	128	—	—
	128	$3 \cdot 6$	—	—
Action	$64 + 64 + 64$	128	—	—
	128	128	—	—
	128	$3 \cdot 6 \cdot 256$	—	—

Table B3: The `deep` architecture.

B.2 OTHER DETAILS

The raw sensory input to the agent is the observed image, in grayscale, without any additional preprocessing. The resolution is 84×84 pixels for the `shallow` model and 160×120 pixels for the `deep` one. We normalized the measurements by their standard deviations under random exploration.

We performed frame skipping during both training and testing. The agent observes the environment and selects an action every 4-th frame. The selected action is repeated during the skipped frames. This accelerates training without sacrificing accuracy. In the paper, “step” always refers to steps after frame skipping (equivalent to every 4-th step before frame skipping). When played by a human, Doom runs at 35 frames per second, so one step of the agent is equivalent to 114 milliseconds of real time. Therefore, frame skipping has the added benefit of bringing the reaction time of the agent closer to that of a human.

We set the temporal offsets τ_1, \dots, τ_n of the predicted future measurements to 1, 2, 4, 8, 16, 32 steps in all experiments. The longest temporal offset corresponds to 3.66 seconds of real time.

We used an experience memory of $M = 20,000$ steps, and sampled a mini-batch of $N = 64$ samples after every $k = 64$ new experiences added. We added the experiences to the memory using 8 copies of the agent running in parallel. The networks in all experiments were trained using the Adam algorithm (Kingma & Ba, 2015) with $\beta_1 = 0.95$, $\beta_2 = 0.999$, and $\varepsilon = 10^{-4}$. The initial learning rate is set to 10^{-4} and is gradually decreased during training. The shallow networks were trained for 800,000 mini-batch iterations (or 51.2 million steps), the deep ones for 1,000,000 iterations, the large one for 2,500,000 iterations.

C BASELINES

We compared our approach to three prior methods: DQN (Mnih et al., 2015), DSR (Kulkarni et al., 2016b), and A3C (Mnih et al., 2016). We used the authors’ implementations of DQN (<https://github.com/kuz/DeepMind-Atari-Deep-Q-Learner>) and DSR (<https://github.com/Ardavans/DSR>), and an independent implementation of A3C (<https://github.com/muupan/async-rl>). For scenarios D1 and D2 we used the change in health as reward. For D3 and D4 we used a linear combination of changes of the three measurements with the same coefficients as for the presented approach: (0.5, 0.5, 1). For DQN we tested three learning rates: the default one (0.00025) and two alternatives (0.00005 and 0.00002). Other DQN hyperparameters were left at their default values. DSR was tested with the default hyperparameters only due to its restrictive running time. For A3C, which trains faster, we performed a search over a set of learning rates ($\{2, 4, 8, 16, 32\} \cdot 10^{-4}$) for the first two tasks; for the last two tasks we trained 20 models with random learning rates sampled log-uniformly between 10^{-4} and 10^{-2} and random β (entropy regularization) sampled log-uniformly between 10^{-4} and 10^{-1} . For all baselines we report the best results we were able to obtain.